I C A S E R E P O R T

PERFORMANCE OF A DATA BASE MANAGEMENT SYSTEM WITH PARTIALLY LOCKED VIRTUAL BUFFERS

Richard S. Brice

Stephen W. Sherman

Report No. 76-6

February 27, 1976

INSTITUTE FOR COMPUTER APPLICATIONS

IN SCIENCE AND ENGINEERING

Operated by the

UNIVERSITIES SPACE RESEARCH ASSOCIATION

at

NASA Langley Research Center

Hampton, Virginia

(NASA-CR-185729) PERFORMANCE OF A DATA BASE MANAGEMENT SYSTEM WITH PARTIALLY LOCKED VIRTUAL BUFFERS (ICASE) 17 p

N89-71336

Unclas

00/60 0224346

PERFORMANCE OF A DATA BASE MANAGEMENT SYSTEM

WITH PARTIALLY LOCKED VIRTUAL BUFFERS

Richard S. Brice

Department of Civil, Mechanical and Environmental Engineering
George Washington University

Stephen W. Sherman*

Institute for Computer Applications in Science and Engineering

ABSTRACT

Buffer pools are created and managed in data base systems in order to reduce the total amount of accesses to the I/O devices. In systems using virtual memory, any reduction in I/O accesses may be accompanied by an increase in paging. In this paper we examine this phenomenon in systems where the virtual buffer is allocated fixed amounts of real memory and partitioned from the program. Our analysis utilizes empirical data gathered in a multifactor experiment. The factors we consider are memory size, virtual buffer size, replacement algorithm for memory, buffer management algorithm, and size of real memory allocated to the buffer.

Acknowledgement

We wish to thank Professor Jim Browne for suggesting this area of research and Professors Browne and Arden for their comments on our initial research. We also wish to acknowledge the technical support provided by Ron Murphy.

* On leave from the University of Houston

This paper was prepared as a result of work performed under NASA Contract No. NAS1-14101 while the second author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665. The work of the first author was supported by NASA Grant NGR-09-010-078.

Introduction

Computer programs that require substantial amounts of I/O often use part of primary memory as a buffer for data from secondary memory. If the overhead to manage the data in primary memory is negligible and the buffer consumes previously unused primary memory, then the use of buffers improves performance due to the faster access to primary than to secondary memory. The use of a buffer in a virtual memory system may cause a decrease in performance due to competition for primary memory between the program and the buffer. Performance can also be degraded by double paging. The dynamics of double paging was characterized by Goldberg and Hassinger [1] as the running of a paged operating system under a paged virtual machine monitor. In this paper, double paging refers to the management of buffer storage under the control of a paged virtual memory environment.

We define virtual buffers as the I/O buffers in a program running on a virtual memory system. In a previous study [2] of the use of virtual buffers, we conducted a multifactor experiment to study the effects of four factors on performance. The factors were virtual buffer manager, virtual buffer size, primary memory size and paging replacement algorithm. This series of 240 experiments was conducted in a controlled laboratory environment [3] by running a data base management program on a dedicated system and measuring the performance as we varied the factors. The data base management program executed a predetermined and unvarying script.

The experiments in the previous study did not partition real memory between the program and its virtual buffer. The program, operating system and virtual buffer, all competed for real memory. The real memory was managed in a global page table by the page replacement algorithm.

In this study we partition memory between the program and the virtual buffer so that the impact of the control variables can be more effectively evaluated. We implement the partition by modifications to the page replacement algorithm. We conduct a set of partitioned experiments in a controlled environment using the same data base management program and script that was used in the non-partitioned experiments. We vary an interesting subset of the same factors used in the non-partitioned experiments and, in addition, vary the amount of real memory allocated to the virtual buffer.

The partitioned experiments show that the least amount of paging in the system is achieved when different paging algorithms are used for the program partition and the buffer partition. Significant variations in performance are produced when the amount of primary memory is fixed and the partition size is varied and when the partition size is also fixed and the virtual buffer size is varied. The best system performance was achieved when the virtual buffer was either the same size as the buffer partition or was much larger than the partition. The partitioned experiments also show that while double paging is a significant factor, the variation in double paging due to the interaction between the paging algorithm and the buffer manager is not significant.

Environment

The partitioned and non-partitioned experiments were conducted on a PRIME 300 minicomputer. The PRIME 300 has a 16-bit word size and supports up to 256K words of real memory (1K=1024 words). Our system has 64K words. The PRIME has virtual memory hardware which supports up to 512 pages of 512 words each. The PRIME peripherals of interest in these experiments consist of two moving head disks each having a capacity of 3 million words.

We have instrumented the PRIME's operating system with a software probe. The probe is locked in memory and cannot interact with the paging process. The probe records events that cause a significant change in the system such as the occurrence of a page fault. The application program we used in our experiments is a prototype data base management (DBM) system. The DBM is run on a dedicated machine with the software probe collecting significant events on tape for later analysis. The DBM system executes the same script of data base requests in each of the experiments. The script completely traverses the data base and causes reading, insertion and deletion of data as it is executed.

The DBM system organizes data in a tree structured format. Requests are made in a series of primitive functions that perform elementary operations on the data base. The data, names and pointers in the data base are encoded into 40-word segments. One physical disk record contains eleven data base segments. The data base used in the experiments contains 45 records. We allocate each disk record to a separate page in the virtual buffer to avoid physical page boundary overlaps. The DBM virtual buffer size is chosen by the user when the DBM system is initiated. The size of the virtual buffer can range from 1 to 64 pages in 1 page increments. A more detailed description of the environment can be found in [2].

Previous Studies

Tuel [4,5] originally studied the paging and I/O performance of an IBM data base system (IMS Version 2.4 system running on VS/2 Release 1.6). Tuel postulated a theoretical model of the buffer I/O (paging in the buffer + I/O accesses). His model and empirical results were reasonably close when only the buffer paging was considered. The early version of IMS used in the study forced the virtual buffer to be searched at every I/O request since a pointer array describing the contents of the buffer was not used. Tuel concluded that

buffer I/O increases with increasing virtual buffer size if the buffer is allowed to page. Therefore the least amount of buffer I/O is achieved by reducing the virtual buffer size to fit the number of pages available in real memory.

Our data base system and later versions of IMS now have a pointer array describing the contents of the virtual buffer. Searching the pointer array causes much less paging activity than the early IMS technique of reading the information from the individual buffers. In a previous paper by Sherman and Brice [2] we assume that searching the pointer array generates no page faults. We develop a very simple theoretical model which predicts total I/O per data base request (T) in the virtual buffer as a function of virtual buffer size in pages (N), pages of real memory allocated to the virtual buffer (M) and the number of pages in the data base (D). The model assumes that the random (RAND) page replacement algorithm and RAND buffer manager are used and the data base requests are uniformly distributed. Total I/O per data base request in the virtual buffer is given by

$$T(M,N,D) = (1- \frac{M}{N}) \text{ page faults} + (1- \frac{N}{D}) \text{ I/O accesses for}$$

$$1 \leq M \leq N \leq D.$$

By use of this model, we show that it is possible to increase the virtual buffer size (N) and reduce the total I/O per data base request in the buffer for various fixed values of M and D.

Although the model assumes that the program and buffer are partitioned, the paging algorithms used in our previous experiments did not partition memory. The factors we considered in the non-partitioned experiments are memory size, virtual buffer size, page replacement algorithm and buffer management algorithm.

Our analysis of the empirical results of the non-partitioned experiments led to the following conclusions: The interaction between the paging algorithm and the buffer algorithm did not have a significant effect on the buffer I/O.

While double paging existed, the double paging rate was not significantly different for any combination of paging algorithm and buffer algorithm. The cost of system I/O (buffer I/O + paging in the program) can decrease when the virtual buffer size is increased but that the amount is very dependent on the amount of real memory available. The performance advantages of virtual buffers can overcome the costs of double paging and the increased program paging resulting from the use of virtual buffers.

Experiments

The multifactor non-partitioned experiments of our previous study are extended in this paper to include 5 levels for the size of the memory partition allocated to the virtual buffer. In our partitioned experiments we use the RAND and SCH (second chance [6] also known as the Multics [7] algorithm) page replacement algorithms. The page replacement algorithms treat each partition separately. Pages in one partition are not considered by the page replacement algorithm when a fault occurs in the other partition. The changes in the page replacement algorithms to allow partitioning are the only differences in these experiments and the previous experiments by Sherman and Brice [2]. The partitioned experiments use the FIFO (first in - first out), RAND, SCH, and LRU (least recently used) virtual buffer managers and virtual buffer sizes of 1, 5, 10, 15, and 20 pages. The total amount of real memory is set to 40K, 44K, and 48K.

The buffer partitions for these experiments are 1, 5, 10, 15, and 20 pages. We only perform experiments for combinations of virtual buffer size and buffer partition where the buffer partition is less than or equal to the virtual buffer size. It would be unreasonable to execute a program with a virtual buffer size smaller than the buffer partition. For example, experiments with a virtual buffer size of 20 pages are combined with all the possible buffer partitions while experiments with a virtual buffer size of 5 are combined with buffer partitions of 1 and 5 pages.

A total of 360 experiments are performed and analyzed. The different experiments are defined by combinations of the 2 paging algorithms, 4 buffer managers, 3 real memory sizes, 5 memory partition sizes and virtual buffer sizes where the virtual buffer was at least as large as the buffer partition size.

Results

Total I/O in the partitioned system (system I/O) is composed of I/O accesses in the virtual buffer, paging in the virtual buffer and paging in the program. The I/O accesses in the virtual buffer are dependent on the virtual buffer size and the buffer management algorithm. The I/O accesses are listed in Table I. Paging in the virtual buffer is dependent on the page replacement algorithm in the buffer, the number of real pages allocated to the buffer (buffer partition), the virtual buffer size and the buffer management algorithm. The paging in the buffer is shown in Table II. Table II divides paging in the buffer into paging caused by the disk I/O manager (double paging) and paging caused by attempts to reference the data in the buffer (reference paging). Paging in the program is a function of the page replacement algorithm in the program partition and the amount of real memory available in the program partition. Table III contains the average program page fault counts for all of the different program partition sizes. A program partition size is determined by subtracting buffer partition size from real memory size. For each of the three real memory sizes there are five buffer partition sizes giving a total of fifteen program partition sizes.

In order to omit congestion we usually present figures that are representative rather than exhaustive in terms of the number of levels of factors available. A complete set of figures for the partitioned experiments can be constructed from the data in Table I, Table II and Table III.

System I/O is shown in Figures I through III. For a given partition size the smallest values for system I/O are obtained when the virtual buffer size is equal either to the partition (this is equivalent to a non-virtual system in the buffer partition), or the maximum virtual buffer size of 20 pages (at this size a significant reduction in the number of I/O accesses has occurred). When program paging is substantial, such as in the experiments using less real memory (Figure I), system I/O decreases as the program partition becomes larger. In the experiments using more real memory (Figure II and Figure III) system I/O also decreases for larger buffer partitions.

In order to understand the performance of total I/O in the system we examine each of its components separately. The program paging for all experiments is shown in Figure IV with the averages in Table III. The variations due to the program paging requirements of the different buffer managers and different seeds in the RAND page replacement algorithm are relatively small. The paging behaves as expected in the program partition. The SCH page replacement algorithm creates fewer page faults than the RAND algorithm and paging increases as the program partition decreases.

Since program paging increases as the buffer partition increases, a decrease in system I/O can only occur if those components of total I/O in the buffer partition (buffer I/O) decrease. Buffer I/O decreases as the buffer partition increases when the virtual buffer size is equal to the partition size since there is no paging in the buffer and the number of I/O access decreases. When the decrease in buffer I/O is greater than the corresponding increase in program paging, system I/O is reduced. Points A, B, C of Figure III illustrate cases where system I/O is reduced by an increase in buffer partition size. Points C, D, E of Figure III show cases where system I/O is increased by an increase in buffer partition size.

Slight decreases in system I/O due to an increased buffer partition size are observed in some of the experiments where total memory size is 48K. Noticeable increases in system I/O are typical when total memory size is 40K. This is illustrated by points A, B, C, D, E of Figure I.

Once a buffer partition is established, the virtual buffer size is allowed to range from the buffer partition size to 20 pages in our experiments. We define the buffer ratio to be virtual buffer size divided by the buffer partition size. A buffer ratio of 1 represents a non-virtual buffer partition (no paging can occur in the buffer). Increasing the buffer ratio decreases disk accesses and increases paging in the buffer. Increasing the buffer ratio can only reduce buffer I/O if the decrease in I/O accesses is greater than the increase in buffer paging.

Buffer I/O is composed of I/O accesses, double paging and reference paging. Double paging occurs when the buffer manager chooses to replace the information in a page that is not in real memory. Reference paging refers to page faults caused by attempts to use information that is in the virtual buffer but not in real memory. We define the double paging rate to be the number of double page faults divided by the number of I/O accesses. We define the reference paging rate to be the number of reference faults divided by 1075 minus the number of I/O accesses. The maximum number of disk accesses which can occur in our experiments is 1075. Table II divides the paging for all the partitioned experiments into reference and double paging. Table I contains the I/O accesses for all of the partitioned experiments. Buffer I/O for a given page replacement algorithm, buffer manager, buffer partition size and virtual buffer size is obtained by adding the corresponding three values from Table I and Table II.

Some observed values for buffer I/O are shown in Figure V. Buffer I/O typically shows a sharp increase as soon as the buffer ratio becomes greater than 1. This is followed by a decrease in buffer I/O as the buffer ratio is

extended to its maximum value. The minimum value for buffer I/O in a given partition is usually observed when the buffer ratio is 1.

The sharp increase in buffer I/O is caused by a sharp increase in paging when the buffer ratio is slightly larger than 1. If the number of I/O access is close to its maximum value of 1075 when the buffer ratio is 1, the increase in paging is due to an increase in double paging. If the number of I/O accesses is small when the buffer ratio is 1, a sharp increase in reference paging causes the increase in paging. This phenomenon can be seen from an examination of Tables I and II. The double paging is high when the I/O accesses are large because the double paging rate increases very quickly as soon as the buffer ratio is greater than 1 for the SCH, FIFO and LRU buffer managers. The RAND buffer manager has a double paging rate that does not increase as quickly as the other buffer managers. Figure VI shows that the double paging rate for the SCH paging algorithm increases close to its maximum value of 1 for all the buffer managers except RAND. The double paging rates for the RAND paging algorithm increase almost as steeply as those shown in Figure VI but terminate at significantly lower values for all the buffer managers except RAND. The reference paging rate increases almost linearly with virtual buffer size and is insensitive to the page replacement algorithm.

Table I shows that the I/O accesses decrease as the virtual buffer size increases and the RAND buffer manager requires fewer I/O access than the others. The better performance of the RAND buffer manager is due to the record reference pattern which contains a few highly referenced records usually separated by strings of references [2].

Figure V shows that the RAND buffer manager has less buffer I/O than the SCH buffer manager. The FIFO and LRU buffer managers produce results similar to the SCH buffer manager. The performance of the RAND buffer manager is better than the others because it typically has less I/O accesses and a

lower double paging rate. This combination results in less buffer I/O even though the reference paging of the RAND buffer manager is higher at virtual buffer sizes of 10 and 15.

An examination of Table II shows that the RAND pager produces slightly fewer page faults than the SCH pager for the RAND buffer manager. The RAND pager often produces significantly fewer page faults than the SCH pager for the other buffer managers. This is a direct reflection of the reduction in the double paging rate by the RAND pager. The best combination of paging algorithm and buffer manager to reduce the buffer I/O to a minimum when the buffer ratio is greater than 1, is clearly the RAND paging algorithm and the RAND buffer manager.

Figure VII contains the buffer I/O for all buffer partition sizes with a combination of RAND page replacement algorithm and RAND buffer manager. Corresponding model predictions using the formula presented in a earlier section for total I/O per data base request are also shown in Figure VII for comparison. In the model the record references were randomly distributed in D with equal probability. We showed in [2] that the distribution of the record references generated by the test bed script more closely approximated an exponential distribution. Approximately 90% of the record references were contained in the most frequently referenced 25 records. The model predictions are based on a data base size (D) of 25 records. As the value of D is increased to 45 (the size of the experimental data base) the values for predicted buffer I/O increase. The model is quite robust considering its simplicity.

The partitioned experiments are conducted with the same page replacement algorithm in each partition. Memory partitioning removes any interaction between program paging and buffer I/O. This allows us to combine the buffer I/O generated in one test case with the program paging generated in another test case partitioned the same way. Since the buffer I/O is minimized by use

of the RAND paging algorithm and the program paging is minimized with the SCH paging algorithm, a combination of these two paging algorithm will produce the minimum system I/O.

We performed an analysis of variance [8] of the values for buffer paging for buffer partition sizes of 1, 5, 10 and 15 pages. The analysis does not include the cases where the buffer ratio is 1. The factors included in the analysis are buffer manager, page replacement algorithm and virtual buffer size. The interaction of the buffer manager and the page replacement algorithm do not cause a significant amount of the variation. The buffer managers caused most of the variation for the small buffer partitions. The buffer managers and paging algorithms both affect the variation significantly for the large buffer partitions.

Conclusion

We show that the size of the buffer partition can cause significant variation in system I/O due to increased program paging. With partition size fixed, increasing the size of a virtual buffer over the partition size increases buffer I/O significantly initially and then buffer I/O decreases. When I/O accesses are considered equal in cost to page faults, the buffer I/O at 20 pages is usually slightly greater than the buffer I/O when the virtual buffer was equal to the partition size. If I/O accesses cost more in terms of time than page faults (as is usually the case in most systems) then the use of virtual buffers can improve system performance.

There is a significant double paging effect in the buffer. The RAND page replacement algorithm produced less paging in the buffer than the SCH algorithm because it produced a lower double paging rate although the SCH algorithm was clearly superior in the program partition. The RAND buffer manager produced less buffer I/O than the SCH, FIFO and LRU buffer managers because it produced

a lower double paging rate and fewer I/O accesses due to the stringed character of the record requests. More variation in buffer I/O is usually due to the buffer managers than the paging algorithm but the interaction of those factors does not cause a significant amount of variation.

Our experiments show that it is possible for the use of a virtual buffer to improve performance in a partitioned system. The chances for improving performance are increased if: (1) The cost of I/O accesses are greater than the cost of paging. (2) The RAND paging algorithm is used in the buffer partition. (3) The SCH paging algorithm is used in the program partition. (4) The RAND buffer manager is chosen. (5) The virtual buffer size is significantly larger than the buffer partition size.

Bibliography

1. Goldberg, R. and R. Hassinger, "The Double Paging Anomaly," Proc. 1974 National Computer Conference, Chicago, May 6-8, 1974.

2. Sherman, S. W. and R. S. Brice, "Performance of a Data Base Manager in a Virtual Memory System," Proceedings SIGMOD International Conference on Management of Data, June, 1976.

3. Schwetman, H. D. and J. C. Browne, "An Experimental Study of Computer System Performance," Proc. National ACM Conference, Boston, Mass., August 1972, pp. 693-703.

4. Tuel, W. G., "An Analysis of Buffer Paging in Virtual Storage Systems," IBM Report RJ 1421, July 1974.

5. Tuel, W. G., "An Analysis of Buffer Paging in Virtual Storage Systems," Proc. Third Texas Conference on Computing Systems, Austin, Texas, November 1974.

6. Hoare, C. A. R. and R. M. McKeag, "A Survey of Store Management Techniques," A.P.I.C. Studies in Data Processing, No. 9, Academic Press, 1972.

7. Corbato, F. J., "A Paging Experiment with the Multics System," In Honor of P. M. Morse, M.I.T. Press, Cambridge, Mass., 1969, pp. 217-228.

8. Tsao, R. F.; L. W. Comeau, and B. H. Margolin, "A Multi-Factor Paging Experiment I, II," Statistical Computer Performance Evaluation, edited by Walter Freiberger, Academic Press, pp. 103-158.

Virtual Buffer Size in Pages	Buffer Managers			
	LRU	SCH	RAND	FIFO
1	1075	1075	1075	1075
5	784	791	774	794
10	684	687	559	688
15	418	422	270	434
20	93	98	154	103

TABLE I. Number of I/O requests to read records into the virtual buffer.

	Pages Allocated To Buffer	Virtual Buffer Size	FIFO		LRU		RAND		SCH	
			Source of Fault							
			DP	RP	DP	RP	DP	RP	DP	RP
RAND PAGER	1	1	1	0	1	0	1	0	1	0
		5	787	279	784	289	625	298	787	282
		10	681	385	684	387	534	511	687	384
		15	434	637	418	653	246	797	422	648
		20	102	965	94	975	152	915	99	970
	5	5	12	4	8	0	10	7	15	3
		10	553	110	590	113	283	220	587	96
		15	408	359	405	374	184	507	404	351
		20	100	670	94	682	126	609	98	687
	10	10	25	5	14	4	21	10	20	5
		15	291	194	275	199	94	271	268	185
		20	82	445	85	448	78	379	87	441
	15	15	41	17	36	13	25	29	39	13
		20	64	200	59	196	42	135	59	178
	20	20	37	44	33	27	21	8	34	29
SCH PAGER	1	1	1	0	1	0	1	0	1	0
		5	786	279	784	289	625	298	787	282
		10	680	385	684	387	534	511	686	384
		15	433	637	418	653	246	797	421	648
		20	101	965	94	975	152	915	98	970
	5	5	5	0	5	0	5	0	5	0
		10	649	101	684	108	300	219	675	103
		15	428	358	418	374	190	512	419	370
		20	98	689	94	698	124	634	98	693
	10	10	10	0	10	0	10	0	10	0
		15	405	250	416	272	200	393	406	270
		20	92	582	93	602	91	529	92	585
	15	15	15	0	15	0	15	0	15	0
		20	88	330	93	330	44	154	88	255
	20	20	20	0	20	0	20	0	20	0

TABLE II. Double paging (DP) and reference paging (RP) in the virtual buffer.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	1	70	1301	[illegible]
[illegible]	5	75	[illegible]	[illegible]
[illegible]	10	70	[illegible]	[illegible]
[illegible]	15	69	[illegible]	[illegible]
[illegible]	20	68	6964	4724
44k	1	87	259	60
44k	5	83	623	358
44k	10	78	1482	1402
44k	15	73	2603	2194
44k	20	68	3825	2935
48k	1	95	187	28
48k	5	91	239	37
48k	10	86	308	96
48k	15	81	949	809
48k	20	76	1927	1858

TABLE III. Average program paging for SCH and RAND page replacement algorithms.

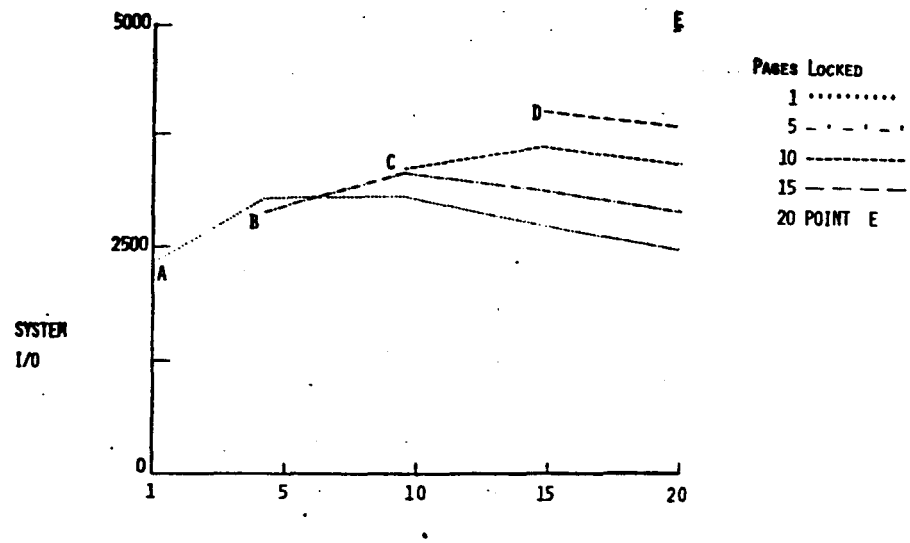

FIGURE I

SYSTEM I/O (PROGRAM PAGING + BUFFER PAGING + I/O ACCESSES) FOR PARTITIONED EXPERIMENTS USING SCH PAGE REPLACEMENT ALGORITHM, RAND BUFFER MANAGER AND 40K MEMORY.
AT POINTS A, B, C, D, E, BUFFER RATIO = 1

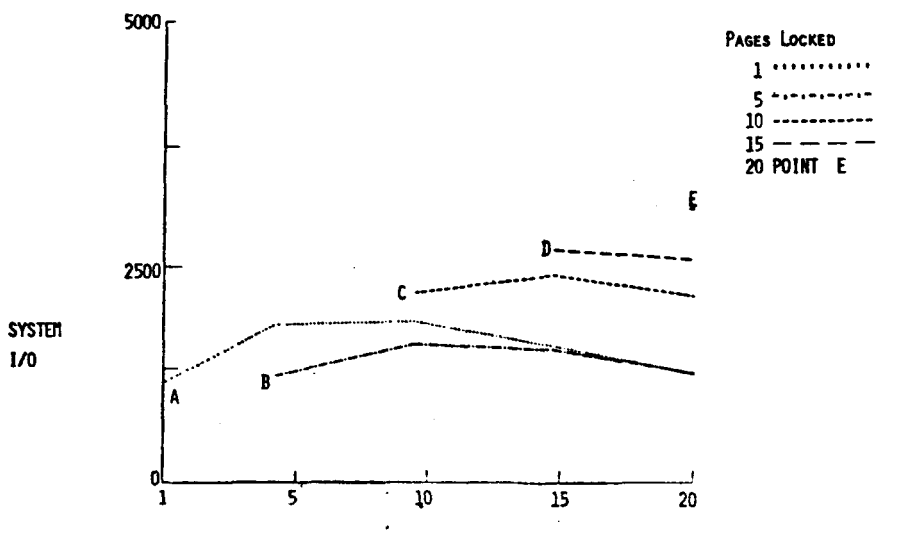

FIGURE II

SYSTEM I/O (PROGRAM PAGING + BUFFER PAGING + I/O ACCESSES) FOR PARTITIONED EXPERIMENTS USING SCH PAGE REPLACEMENT ALGORITHM, RAND BUFFER MANAGER AND 44K MEMORY.
AT POINTS A, B, C, D, E, BUFFER RATIO = 1.

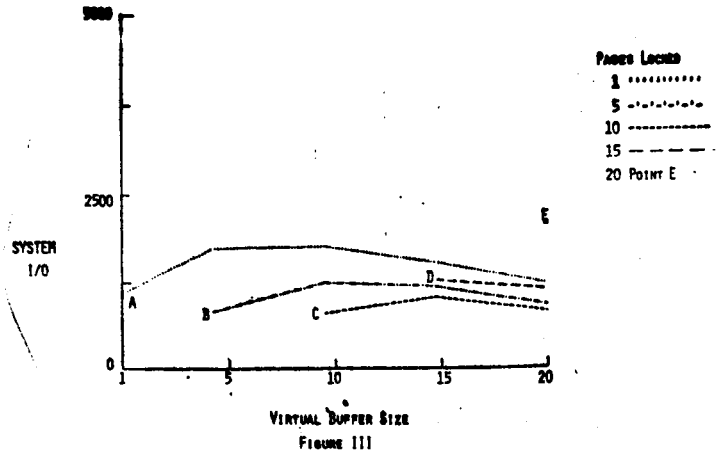

Figure III

System I/O (Program Paging + Buffer Paging + I/O Accesses) for partitioned experiments using SCH page replacement algorithm, RAND buffer manager and 48K memory.
At Points A, B, C, D, E, Buffer Ratio = 1.

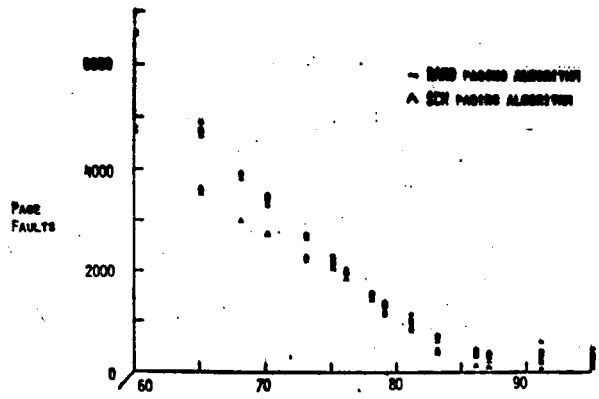

Figure IV

Program paging for SCH and RAND page replacement algorithms

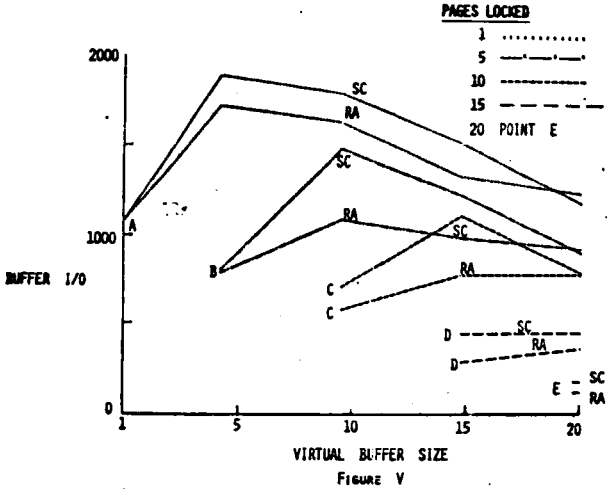

Figure V

Buffer I/O (Buffer Paging + I/O Accesses) for combination of SCH page replacement algorithm with RAND buffer manager (RA) and SCH buffer Manager (SC).

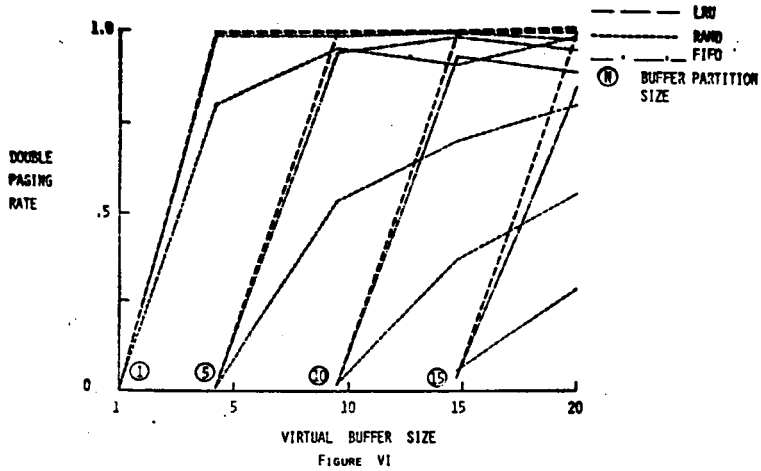

Figure VI

Double Paging Rate for SCH Page Replacement Algorithm with FIFO, LRU, and RAND Buffer Managers

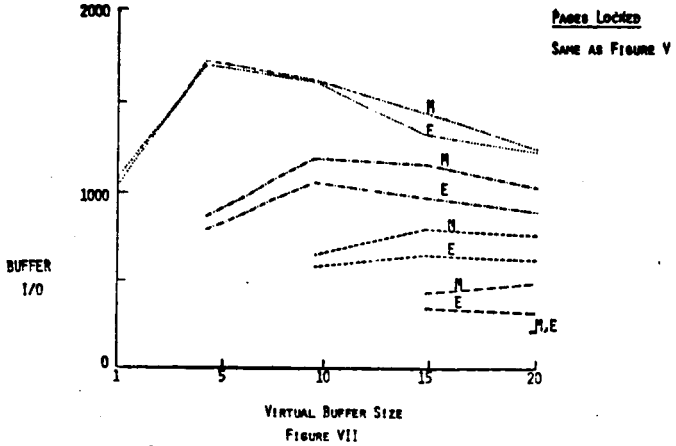

Figure VII

Buffer I/O (Paging in the buffer + I/O Accesses) for the model (M) and the partitioned experiment (E) with RAND page replacement algorithm and RAND buffer manager.